Effective Compilation of Higher-Order Programs

Roland Leißa Klaas Boesche Sebastian Hack
Richard Membarth Arsène Pérard-Gayot Philipp Slusallek

`http://compilers.cs.uni-saarland.de`

`https://github.com/AnyDSL/thorin`

Compiler Design Lab
Saarland University

SIC Saarland Informatics Campus

Introduction

Intermediate Representations (IRs)

imperative languages C, Fortran, ...

- instruction lists + CFGs
  - LLVM
  - GIMPLE (gcc)

Intermediate Representations (IRs)

imperative languages C, Fortran, ...

- instruction lists + CFGs
  - LLVM
  - GIMPLE (gcc)
- graph-based IRs – “sea of nodes” [Click95]
  - Java Hotspot
  - libFirm
  - TurboFan (Google’s JavaScript compiler)

Intermediate Representations (IRs)

imperative languages C, Fortran, ...

- instruction lists + CFGs
  - LLVM
  - GIMPLE (gcc)
- graph-based IRs – “sea of nodes” [Click95]
  - Java Hotspot
  - libFirm
  - TurboFan (Google’s JavaScript compiler)

functional languages Haskell, ML, ...

- λ-calculus
  - Core (GHC)
  - Lambda IR (OCaml)
  - Continuation Passing Style (CPS) [Appel06]

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

Motivation: Post-Order Visit

```
struct Node {
    int data;
    Node* left;
    Node* right;
};

void post_order_visit(Node* n) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    cout << n->data << endl;
}
```

How to factor-out the Visiting Algorithm?

Two Choices:

How to factor-out the Visiting Algorithm?

Two Choices:

1. Iterators
   - standard conforming iterators: expert C++ knowledge
   - additional pointers in `Node` or
   - explicit, heap-managed stack

How to factor-out the Visiting Algorithm?

Two Choices:

1. Iterators
   - standard conforming iterators: expert C++ knowledge
   - additional pointers in `Node` or
   - explicit, heap-managed stack
2. Higher-order Functions

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Factor-out Visiting Algorithm

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}

void sum(Node* n) {
    int result = 0;
    post_order_visit(n, [&](int d) { result += d; });
    cout << result << endl;
}
```

Compiling

```
void post_order_visit(Node* n, function<void(int)> f) {
    if (n->left)
        post_order_visit(n->left, f);

    if (n->right)
        post_order_visit(n->right, f);

    f(n->data);
}

void print(Node* n) {
    post_order_visit(n, [](int d) { cout << d << endl; });
}
```

```
clang -O3 -fno-exceptions
```

post_order_visit with clang -O3 -fno-exceptions

```
define internal fastcc void @_ZL16post_order_visitP4NodeSt8functionIFviEE(%struct.Node* nocapture readonly %n, %"class.std::function"* %f) unnamed_addr #3 {
entry:
  %__args.addr.i = alloca i32, align 4
  %agg.tmp = alloca %"class.std::function", align 8
  %agg.tmp5 = alloca %"class.std::function", align 8
  %left = getelementptr inbounds %struct.Node, %struct.Node* %n, i64 0, i32 1
  %0 = load %struct.Node*, %struct.Node** %left, align 8, !tbaa !8
  %tobool = icmp eq %struct.Node* %0, null
  br i1 %tobool, label %if.end, label %if.then

if.then:                                          ; preds = %entry
  %_M_manager.i.i = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp, i64 0, i32 0, i32 1
  store i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* null, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i, align 8, !tbaa !6
  %_M_manager.i.i.i = getelementptr inbounds %"class.std::function", %"class.std::function"* %f, i64 0, i32 0, i32 1
  %1 = load i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)*, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i.i, align 8, !tbaa !6
  %lnot.i.i = icmp eq i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* %1, null
  br i1 %lnot.i.i, label %_ZNSt8functionIFviEEC2ERKS1_.exit, label %if.then.i

if.then.i:                                        ; preds = %if.then
  %_M_functor.i = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp, i64 0, i32 0, i32 0
  %_M_functor2.i = getelementptr inbounds %"class.std::function", %"class.std::function"* %f, i64 0, i32 0, i32 0
  %call3.i = call zeroext i1 %1(%"union.std::_Any_data"* dereferenceable(16) %_M_functor.i, %"union.std::_Any_data"* dereferenceable(16) %_M_functor2.i, i32 2) #2
  %2 = bitcast i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i.i to <2 x i64>*
  %3 = load <2 x i64>, <2 x i64>* %2, align 8, !tbaa !11
  %4 = bitcast i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i to <2 x i64>*
  store <2 x i64> %3, <2 x i64>* %4, align 8, !tbaa !11
  br label %_ZNSt8functionIFviEEC2ERKS1_.exit

_ZNSt8functionIFviEEC2ERKS1_.exit:                ; preds = %if.then, %if.then.i
  call fastcc void @_ZL16post_order_visitP4NodeSt8functionIFviEE(%struct.Node* nonnull %0, %"class.std::function"* nonnull %agg.tmp)
  %5 = load i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)*, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i, align 8, !tbaa !6
  %tobool.i = icmp eq i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* %5, null
  br i1 %tobool.i, label %if.end, label %if.then.i17

if.then.i17:                                      ; preds = %_ZNSt8functionIFviEEC2ERKS1_.exit
  %_M_functor.i16 = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp, i64 0, i32 0, i32 0
  %call.i = call zeroext i1 %5(%"union.std::_Any_data"* dereferenceable(16) %_M_functor.i16, %"union.std::_Any_data"* dereferenceable(16) %_M_functor.i16, i32 3) #2
  br label %if.end

if.end:                                           ; preds = %if.then.i17, %_ZNSt8functionIFviEEC2ERKS1_.exit, %entry
  %right = getelementptr inbounds %struct.Node, %struct.Node* %n, i64 0, i32 2
  %6 = load %struct.Node*, %struct.Node** %right, align 8, !tbaa !12
  %tobool2 = icmp eq %struct.Node* %6, null
  br i1 %tobool2, label %if.end.if.end6_crit_edge, label %if.then3

if.end.if.end6_crit_edge:                         ; preds = %if.end
  %.pre = getelementptr inbounds %"class.std::function", %"class.std::function"* %f, i64 0, i32 0, i32 1
  br label %if.end6

if.then3:                                         ; preds = %if.end
  %_M_manager.i.i18 = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp5, i64 0, i32 0, i32 1
  store i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* null, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i18, align 8, !tbaa !6
  %_M_manager.i.i.i19 = getelementptr inbounds %"class.std::function", %"class.std::function"* %f, i64 0, i32 0, i32 1
  %7 = load i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)*, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i.i19, align 8, !tbaa !6
  %lnot.i.i20 = icmp eq i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* %7, null
  br i1 %lnot.i.i20, label %_ZNSt8functionIFviEEC2ERKS1_.exit27, label %if.then.i26

if.then.i26:                                      ; preds = %if.then3
  %_M_functor.i21 = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp5, i64 0, i32 0, i32 0
  %_M_functor2.i22 = getelementptr inbounds %"class.std::function", %"class.std::function"* %f, i64 0, i32 0, i32 0
  %call3.i23 = call zeroext i1 %7(%"union.std::_Any_data"* dereferenceable(16) %_M_functor.i21, %"union.std::_Any_data"* dereferenceable(16) %_M_functor2.i22, i32 2) #2
  %8 = bitcast i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i.i19 to <2 x i64>*
  %9 = load <2 x i64>, <2 x i64>* %8, align 8, !tbaa !11
  %10 = bitcast i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i18 to <2 x i64>*
  store <2 x i64> %9, <2 x i64>* %10, align 8, !tbaa !11
  br label %_ZNSt8functionIFviEEC2ERKS1_.exit27

_ZNSt8functionIFviEEC2ERKS1_.exit27:              ; preds = %if.then3, %if.then.i26
  call fastcc void @_ZL16post_order_visitP4NodeSt8functionIFviEE(%struct.Node* nonnull %6, %"class.std::function"* nonnull %agg.tmp5)
  %11 = load i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)*, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i18, align 8, !tbaa !6
  %tobool.i29 = icmp eq i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* %11, null
  br i1 %tobool.i29, label %if.end6, label %if.then.i32

if.then.i32:                                      ; preds = %_ZNSt8functionIFviEEC2ERKS1_.exit27
  %_M_functor.i30 = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp5, i64 0, i32 0, i32 0
  %call.i31 = call zeroext i1 %11(%"union.std::_Any_data"* dereferenceable(16) %_M_functor.i30, %"union.std::_Any_data"* dereferenceable(16) %_M_functor.i30, i32 3) #2
  br label %if.end6

if.end6:                                          ; preds = %if.end.if.end6_crit_edge, %if.then.i32, %_ZNSt8functionIFviEEC2ERKS1_.exit27
  %_M_manager.i.i11.pre_phi = phi i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** [ %.pre, %if.end.if.end6_crit_edge ], [ %_M_manager.i.i.i19, %if.then.i32 ], [ %_M_manager.i.i.i19, %_ZNSt8functionIFviEEC2ERKS1_.exit27 ]
  %data = getelementptr inbounds %struct.Node, %struct.Node* %n, i64 0, i32 0
  %12 = load i32, i32* %data, align 8, !tbaa !13
  %13 = bitcast i32* %__args.addr.i to i8*
  call void @llvm.lifetime.start(i64 4, i8* %13)
  store i32 %12, i32* %__args.addr.i, align 4, !tbaa !14
  %14 = load i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)*, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i11.pre_phi, align 8, !tbaa !6
  %lnot.i.i12 = icmp eq i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* %14, null
  br i1 %lnot.i.i12, label %if.then.i13, label %_ZNKSt8functionIFviEEclEi.exit

if.then.i13:                                      ; preds = %if.end6
  call void @_ZSt25__throw_bad_function_callv() #7
  unreachable

_ZNKSt8functionIFviEEclEi.exit:                   ; preds = %if.end6
  %_M_invoker.i14 = getelementptr inbounds %"class.std::function", %"class.std::function"* %f, i64 0, i32 1
  %15 = load void (%"union.std::_Any_data"*, i32)*, void (%"union.std::_Any_data"*, i32)** %_M_invoker.i14, align 8, !tbaa !1
  %_M_functor.i15 = getelementptr inbounds %"class.std::function", %"class.std::function"* %f, i64 0, i32 0, i32 0
  call void %15(%"union.std::_Any_data"* dereferenceable(16) %_M_functor.i15, i32* nonnull dereferenceable(4) %__args.addr.i) #2
  call void @llvm.lifetime.end(i64 4, i8* %13)
  ret void
}
```

print with clang -O3 -fno-exceptions

```
define void @_Z5printP4Node(%struct.Node* nocapture readonly %n) #3 {
entry:
  %agg.tmp = alloca %"class.std::function", align 8
  %_M_manager.i.i = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp, i64 0, i32 0, i32 1
  %_M_invoker.i = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp, i64 0, i32 1
  store void (%"union.std::_Any_data"*, i32*)* @"_ZNSt17_Function_handlerIFviEZ5printP4NodeE3$_0E9_M_invokeERKSt9_Any_dataOi", void (%"union.std::_Any_data"*, i32*)** %_M_invoker.i, align 8, !tbaa !1
  store i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* @"_ZNSt14_Function_base13_Base_managerIZ5printP4NodeE3$_0E10_M_managerERSt9_Any_dataRKS5_St18_Manager_operation", i1 (%"union.std::_Any_data"*,
          %"union.std::_Any_data"*, i32)** %_M_manager.i.i, align 8, !tbaa !6
  call fastcc void @_ZL16post_order_visitP4NodeSt8functionIFviEE(%struct.Node* %n, %"class.std::function"* nonnull %agg.tmp)
  %0 = load i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)*, i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)** %_M_manager.i.i, align 8, !tbaa !6
  %tobool.i = icmp eq i1 (%"union.std::_Any_data"*, %"union.std::_Any_data"*, i32)* %0, null
  br i1 %tobool.i, label %_ZNSt14_Function_baseD2Ev.exit, label %if.then.i

if.then.i:                                        ; preds = %entry
  %_M_functor.i = getelementptr inbounds %"class.std::function", %"class.std::function"* %agg.tmp, i64 0, i32 0, i32 0
  %call.i = call zeroext i1 %0(%"union.std::_Any_data"* dereferenceable(16) %_M_functor.i, %"union.std::_Any_data"* dereferenceable(16) %_M_functor.i, i32 3) #2
  br label %_ZNSt14_Function_baseD2Ev.exit

_ZNSt14_Function_baseD2Ev.exit:                   ; preds = %entry, %if.then.i
  ret void
}

define internal void @"_ZNSt17_Function_handlerIFviEZ5printP4NodeE3$_0E9_M_invokeERKSt9_Any_dataOi"(%"union.std::_Any_data"* nocapture readnone dereferenceable(16) %__functor, i32* nocapture readonly dereferenceable(4) %__args) #3 align 2 {
entry:
  %0 = load i32, i32* %__args, align 4, !tbaa !14
  %call.i = tail call dereferenceable(272) %"class.std::basic_ostream"* @_ZNSolsEi(%"class.std::basic_ostream"* nonnull @_ZSt4cout, i32 %0) #2
  %1 = bitcast %"class.std::basic_ostream"* %call.i to i8**
  %vtable.i.i = load i8*, i8** %1, align 8, !tbaa !15
  %vbase.offset.ptr.i.i = getelementptr i8, i8* %vtable.i.i, i64 -24
  %2 = bitcast i8* %vbase.offset.ptr.i.i to i64*
  %vbase.offset.i.i = load i64, i64* %2, align 8
  %3 = bitcast %"class.std::basic_ostream"* %call.i to i8*
  %add.ptr.i.i = getelementptr inbounds i8, i8* %3, i64 %vbase.offset.i.i
  %_M_ctype.i.i = getelementptr inbounds i8, i8* %add.ptr.i.i, i64 240
  %4 = bitcast i8* %_M_ctype.i.i to %"class.std::ctype"**
  %5 = load %"class.std::ctype"*, %"class.std::ctype"** %4, align 8, !tbaa !17
  %tobool.i5.i = icmp eq %"class.std::ctype"* %5, null
  br i1 %tobool.i5.i, label %if.then.i6.i, label %_ZSt13__check_facetISt5ctypeIcEERKT_PS3_.exit.i

if.then.i6.i:                                     ; preds = %entry
  tail call void @_ZSt16__throw_bad_castv() #7
  unreachable

_ZSt13__check_facetISt5ctypeIcEERKT_PS3_.exit.i:  ; preds = %entry
  %_M_widen_ok.i.i = getelementptr inbounds %"class.std::ctype", %"class.std::ctype"* %5, i64 0, i32 8
  %6 = load i8, i8* %_M_widen_ok.i.i, align 8, !tbaa !20
  %tobool.i.i = icmp eq i8 %6, 0
  br i1 %tobool.i.i, label %if.end.i.i, label %if.then.i.i

if.then.i.i:                                      ; preds = %_ZSt13__check_facetISt5ctypeIcEERKT_PS3_.exit.i
  %arrayidx.i.i = getelementptr inbounds %"class.std::ctype", %"class.std::ctype"* %5, i64 0, i32 9, i64 10
  %7 = load i8, i8* %arrayidx.i.i, align 1, !tbaa !22
  br label %"_ZZ5printP4NodeENK3$_0clEi.exit"

if.end.i.i:                                       ; preds = %_ZSt13__check_facetISt5ctypeIcEERKT_PS3_.exit.i
  tail call void @_ZNKSt5ctypeIcE13_M_widen_initEv(%"class.std::ctype"* nonnull %5) #2
  %8 = bitcast %"class.std::ctype"* %5 to i8 (%"class.std::ctype"*, i8)***
  %vtable.i3.i = load i8 (%"class.std::ctype"*, i8)**, i8 (%"class.std::ctype"*, i8)*** %8, align 8, !tbaa !15
  %vfn.i.i = getelementptr inbounds i8 (%"class.std::ctype"*, i8)*, i8 (%"class.std::ctype"*, i8)** %vtable.i3.i, i64 6
  %9 = load i8 (%"class.std::ctype"*, i8)*, i8 (%"class.std::ctype"*, i8)** %vfn.i.i, align 8
  %call.i4.i = tail call signext i8 %9(%"class.std::ctype"* nonnull %5, i8 signext 10) #2
  br label %"_ZZ5printP4NodeENK3$_0clEi.exit"

"_ZZ5printP4NodeENK3$_0clEi.exit":                ; preds = %if.then.i.i, %if.end.i.i
  %retval.0.i.i = phi i8 [ %7, %if.then.i.i ], [ %call.i4.i, %if.end.i.i ]
  %call1.i.i = tail call dereferenceable(272) %"class.std::basic_ostream"* @_ZNSo3putEc(%"class.std::basic_ostream"* nonnull %call.i, i8 signext %retval.0.i.i) #2
  %call.i.i = tail call dereferenceable(272) %"class.std::basic_ostream"* @_ZNSo5flushEv(%"class.std::basic_ostream"* nonnull %call1.i.i) #2
  ret void
}

define internal zeroext i1 @"_ZNSt14_Function_base13_Base_managerIZ5printP4NodeE3$_0E10_M_managerERSt9_Any_dataRKS5_St18_Manager_operation"(%"union.std::_Any_data"* nocapture dereferenceable(16) %__dest,
                                                                                                                                          %"union.std::_Any_data"* dereferenceable(16) %__source, i32 %__op) #5 align 2 {
entry:
  switch i32 %__op, label %sw.epilog [
    i32 0, label %sw.bb
    i32 1, label %sw.bb1
  ]

sw.bb:                                            ; preds = %entry
  %0 = bitcast %"union.std::_Any_data"* %__dest to %"class.std::type_info"**
  store %"class.std::type_info"* bitcast ({ i8*, i8* }* @"_ZTIZ5printP4NodeE3$_0" to %"class.std::type_info"*), %"class.std::type_info"** %0, align 8, !tbaa !11
  br label %sw.epilog

sw.bb1:                                           ; preds = %entry
  %1 = bitcast %"union.std::_Any_data"* %__dest to %"union.std::_Any_data"**
  store %"union.std::_Any_data"* %__source, %"union.std::_Any_data"** %1, align 8, !tbaa !11
  br label %sw.epilog

sw.epilog:                                        ; preds = %entry, %sw.bb1, %sw.bb
  ret i1 false
}
```

Working with higher-order Functions

- *A Graph-Based Higher-Order Intermediate Representation*
  Leißa, Köster, and Hack.
  CGO 2015

Working with higher-order Functions

- *A Graph-Based Higher-Order Intermediate Representation*
  Leißa, Köster, and Hack.
  CGO 2015
- *Shallow Embedding of DSLs via Online Partial Evaluation*
  Leißa, Boesche, Hack, Membarth, and Slusallek.
  GPCE 2015.

Closure Conversion

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

Closure Conversion

```
void range(int a, int b,
       function<void(int)> f)
{
    if (a < b) {
        f(a);
        range(a+1, b, f);
    }
}

void foo(int n) {
    range(0, n, [=] (int i) {
        use(i, n);
    });
}
```

```
struct closurebase {
    void (*f)(void* c, int i);
};

struct closure {
    closurebase base;
    int n;
};

void lambda(void* c, int i) {
    use(i, (closure* c)->n);
}

void range(int a, int b, void* c) {
    if (a < b) {
        ((closurebase*) c)->f(c, a);
        range(a+1, b, c);
    }
}

void foo(int n) {
    closure c = {{&lambda}, n};
    range(0, n, &c);
}
```

What does LLVM do?

- inline the call to the closure's function pointer

What does LLVM do?

- inline the call to the closure's function pointer
- SSA-construct the closure struct
- dissolve the `struct` to scalar values
  (Scalar Replacement of Aggregates)

What does LLVM do?

- inline the call to the closure's function pointer
- SSA-construct the closure struct
- dissolve the `struct` to scalar values
  (Scalar Replacement of Aggregates)
- usually works well for typical STL algorithms

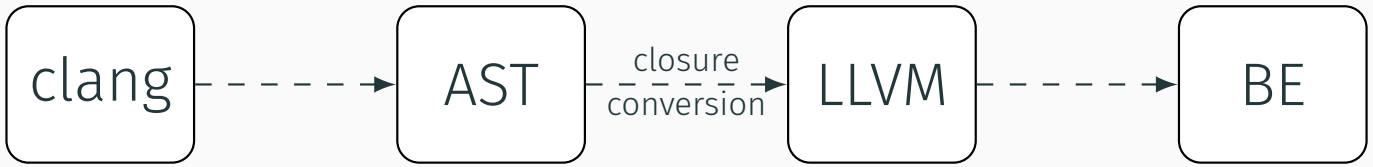
What does LLVM do?

- inline the call to the closure's function pointer
- SSA-construct the closure struct
- dissolve the `struct` to scalar values
  (Scalar Replacement of Aggregates)
- usually works well for typical STL algorithms
- fails for recursive higher-order functions like

What does LLVM do?

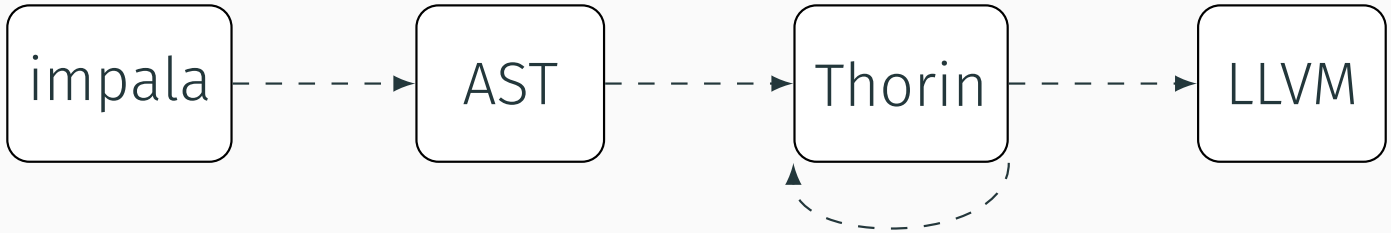
- inline the call to the closure's function pointer
- SSA-construct the closure struct
- dissolve the `struct` to scalar values
  (Scalar Replacement of Aggregates)
- usually works well for typical STL algorithms
- fails for recursive higher-order functions like
  - `range`
  - `post_order_visit`

Closure Conversion



- reimplement for every front-end
- taints the IR with implementation of higher-order functions
  - bloats the IR
  - set of finely tuned analyses & transformations needed for optimization

Closure Conversion



- Thorin = higher-order + CPS + ”sea of nodes”
- directly translate higher-order functions and calls to Thorin
- keep higher-order functions till late during compilation
- powerful closure-elimination phase

Thorin

SSA-Form

```
int foo(int n) {
    int a;
    if (n==0) {
        a = 23;
    } else {
        a = 42;
    }

    return a;
}
```

SSA-Form

```
int foo(int n) {
    int a;
    if (n==0) {
        a = 23;
    } else {
        a = 42;
    }

    return a;
}
```

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = φ(23 [then], 42 [else]);
  return a;
}
```

CPS

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = φ(23 [then], 42 [else]);
  return a;
}
```

CPS

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = φ(23 [then], 42 [else]);
  return a;
}
```

```
foo(n: int, ret: int → ⊥) → ⊥:
  let
    then() → ⊥:
      next(23)
    else() → ⊥:
      next(42)
    next(a: int) → ⊥:
      ret(a)
  in
    branch(n==0, then, else)
```

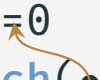
Thorin

```
foo(n: int, ret: int → ⊥) → ⊥:
  let
    then() → ⊥:
      next(23)
    else() → ⊥:
      next(42)
    next(a: int) → ⊥:
      ret(a)
  in
    branch(n==0, then, else)
```

Thorin

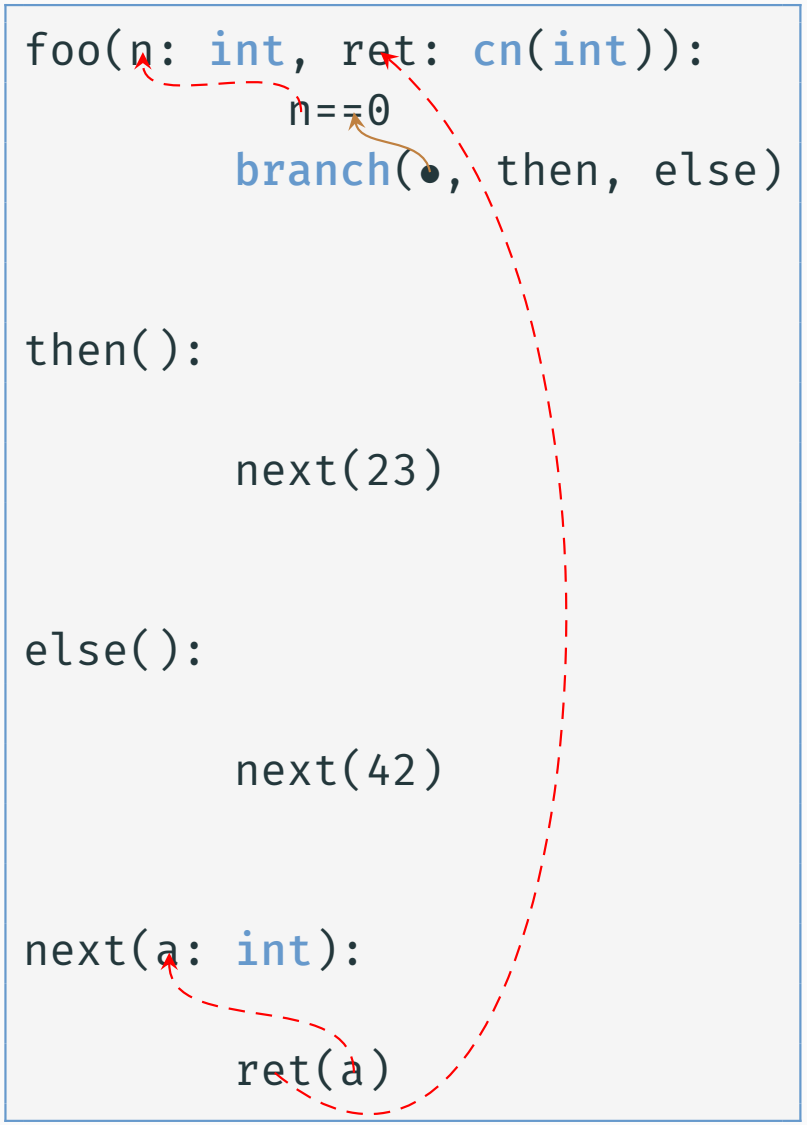
```
foo(n: int, ret: int → ⊥) → ⊥:
  let
    then() → ⊥:
      next(23)
    else() → ⊥:
      next(42)
    next(a: int) → ⊥:
      ret(a)
  in
    branch(n==0, then, else)
```

```
foo(n: int, ret: cn(int)):
          n==0
        branch(•, then, else)


then():

        next(23)


else():

        next(42)


next(a: int):

        ret(a)
```



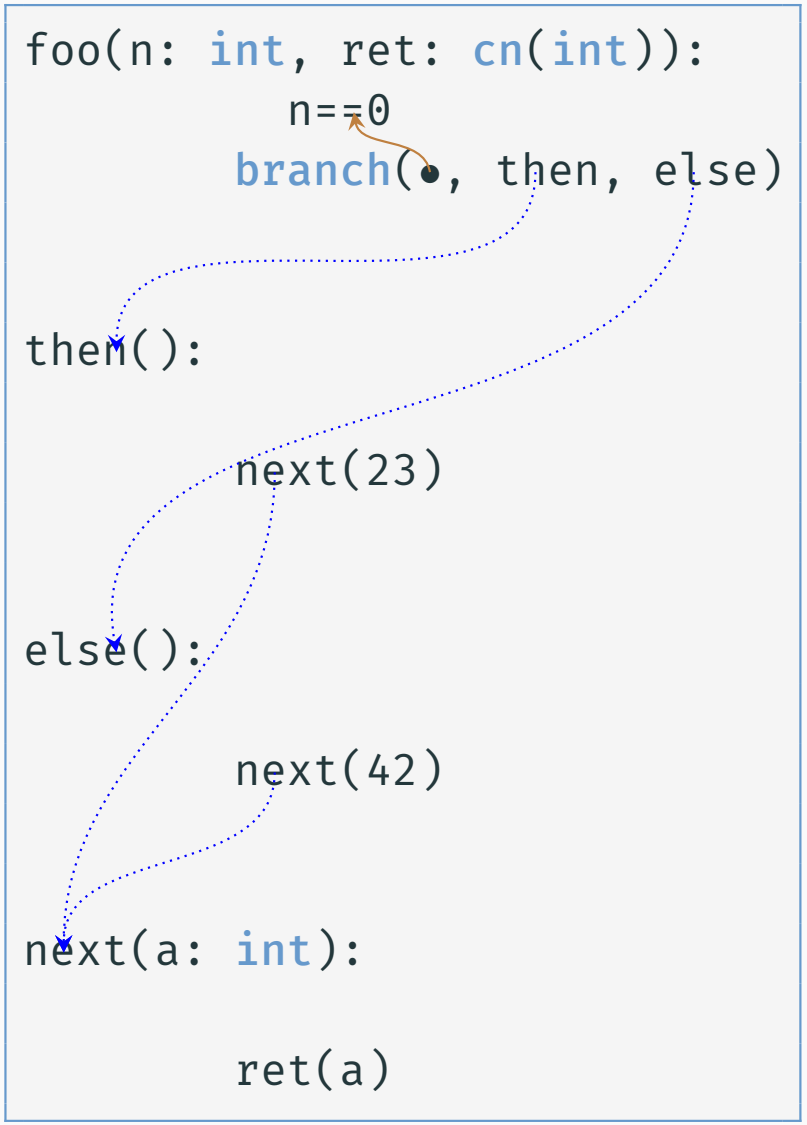
Thorin

```
foo(n: int, ret: int → ⊥) → ⊥:
  let
    then() → ⊥:
      next(23)
    else() → ⊥:
      next(42)
    next(a: int) → ⊥:
      ret(a)
  in
    branch(n==0, then, else)
```



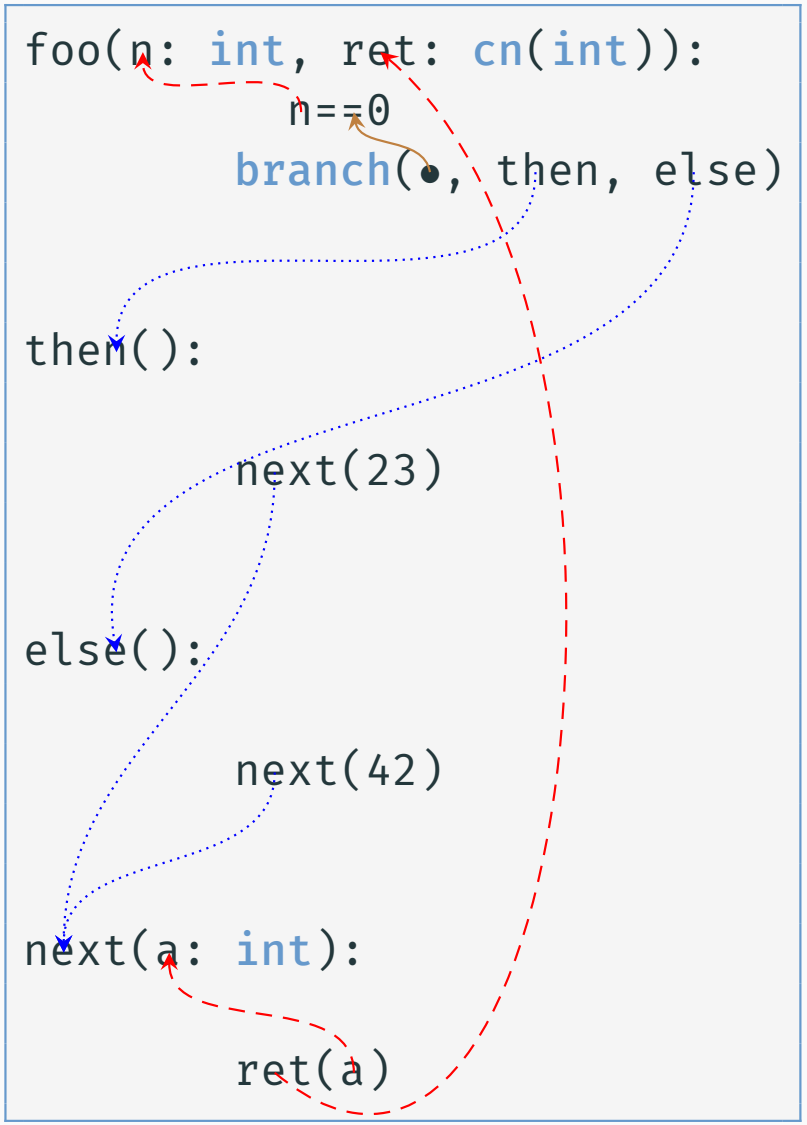
Thorin

```
foo(n: int, ret: int → ⊥) → ⊥:
  let
    then() → ⊥:
      next(23)
    else() → ⊥:
      next(42)
    next(a: int) → ⊥:
      ret(a)
  in
    branch(n==0, then, else)
```



Thorin

```
foo(n: int, ret: int → ⊥) → ⊥:
  let
    then() → ⊥:
      next(23)
    else() → ⊥:
      next(42)
    next(a: int) → ⊥:
      ret(a)
  in
    branch(n==0, then, else)
```



Classic CPS vs Thorin

Classic CPS	Thorin

Classic CPS vs Thorin

Classic CPS	Thorin
let	graph edge (acyclic graph)
letrec	graph edge (cyclic graph)

Classic CPS vs Thorin

Classic CPS	Thorin
let	graph edge (acyclic graph)
letrec	graph edge (cyclic graph)
block nesting	implicit
name resolution	graph edge
name capture	-

SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

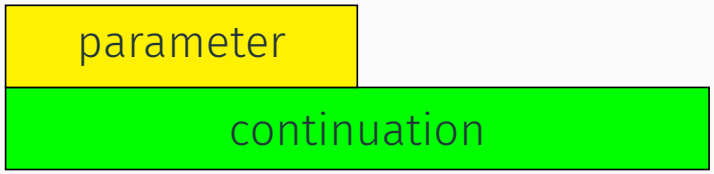
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```

continuation

SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

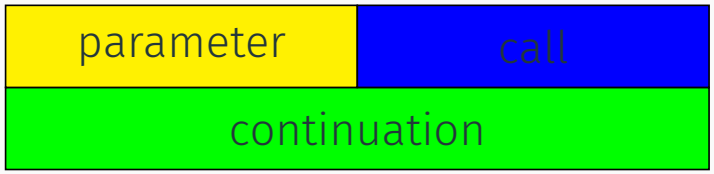
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

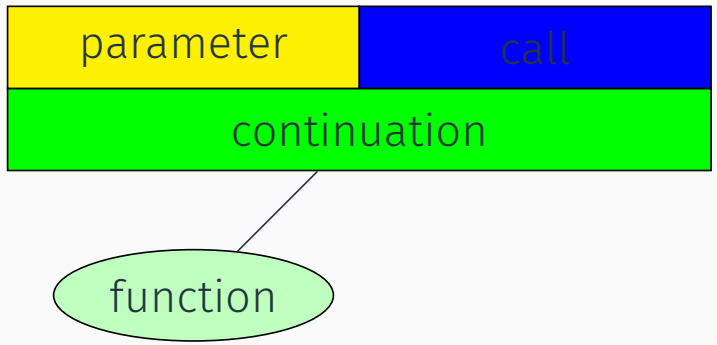
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

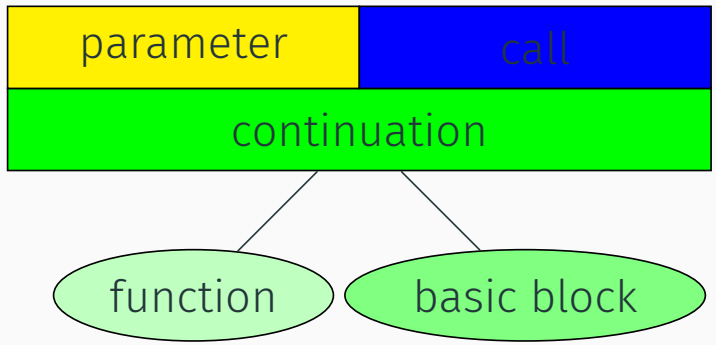
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

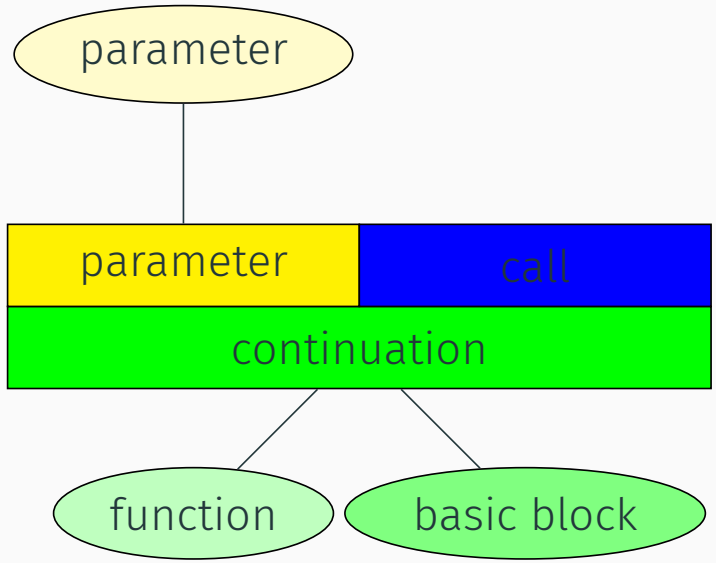
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

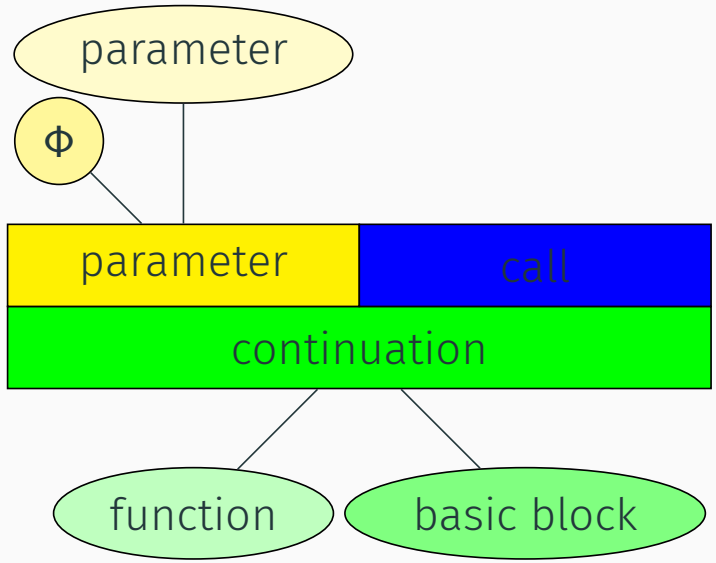
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

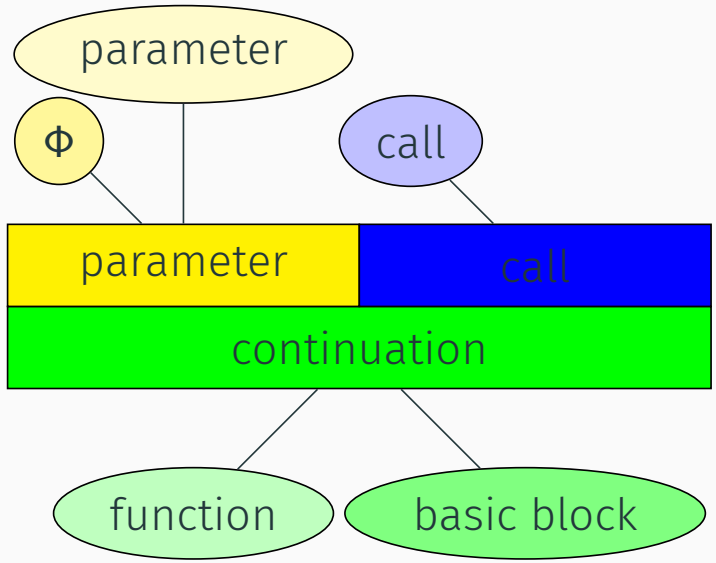
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

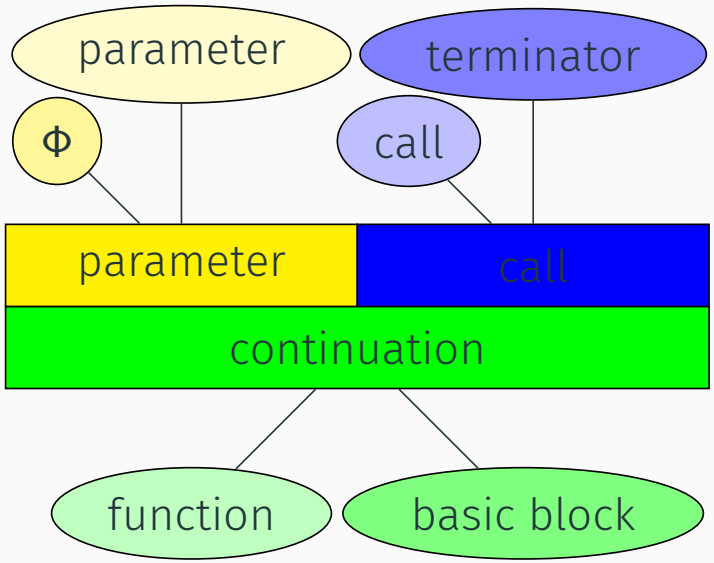
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

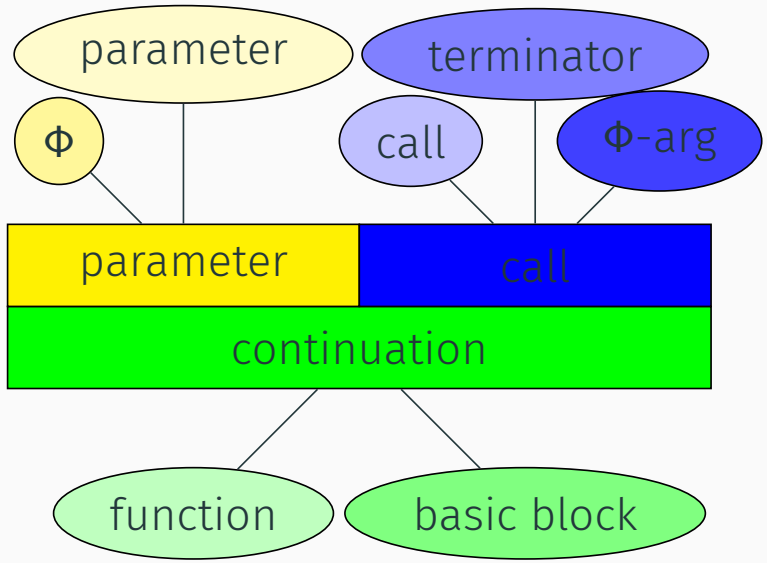
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

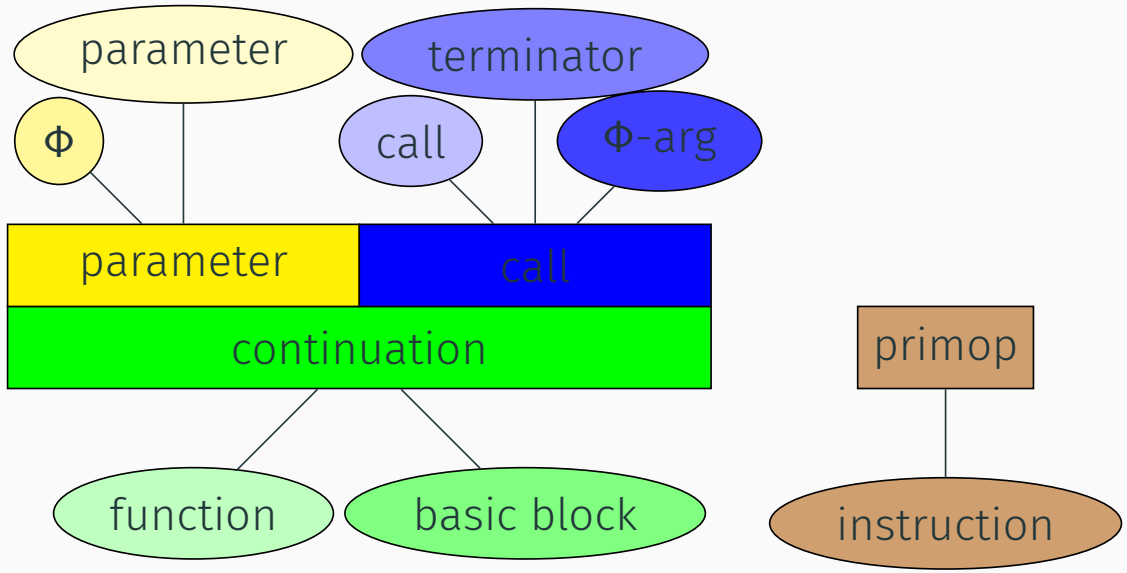
```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



SSA vs Thorin

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```



Lambda Mangling

Control-Flow Form

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = ϕ(23 [then], 42 [else]);
  return a;
}
```

```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```

- Thorin program in CFF if

Control-Flow Form

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = φ(23 [then], 42 [else]);
  return a;
}
```

```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```

- Thorin program in CFF if
  - first-order continuation ⇒ basic block

Control-Flow Form

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = φ(23 [then], 42 [else]);
  return a;
}
```

```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```

- Thorin program in CFF if
  - first-order continuation ⇒ basic block
  - top-level, continuation with “return” ⇒ function

Control-Flow Form

```
int foo(int n) {
  branch(n==0, then, else)
then:
  goto next;
else:
  goto next;
next:
  int a = φ(23 [then], 42 [else]);
  return a;
}
```

```
foo(n: int, ret: cn(int)):
  branch(n==0, then, else)
then():
  next(23)
else():
  next(42)
next(a: int):
  ret(a)
```

- Thorin program in CFF if
  - first-order continuation ⇒ basic block
  - top-level, continuation with “return” ⇒ function
- straightforward to translate to SSA form [Kelsey95]
- no closures needed

Not in CFF

```
void range(int a, int b, function<void(int)> f) {
    //...
    range(a+1, b, f);
}
```

```
range(a: int, b: int, f: cn(int, cn()), ret: cn()):
    /*
     * ...
     */
    range(a+1, b, f, ret)
```

CFF-convertible if

- recursion-free or
- tail-recursive

Not in CFF

```
void range(int a, int b, function<void(int)> f) {
    //...
    range(a+1, b, f);
}
```

```
range(a: int, b: int, f: cn(int, cn()), ret: cn()):
    /*
     * ...
     */
    range(a+1, b, f, ret)
```

CFF-convertible if

- recursion-free or
- tail-recursive

Not in CFF

```
void range(int a, int b, function<void(int)> f) {
    //...
    range(a+1, b, f);
}
```

```
range(a: int, b: int, f: cn(int, cn()), ret: cn()):
    /*
     * ...
     */
    range(a+1, b, f, ret)
```

CFF-convertible if

- recursion-free or
- tail-recursive

Not in CFF

```
void range(int a, int b, function<void(int)> f) {
    //...
    range(a+1, b, f);
}
```

```
range(a: int, b: int, f: cn(int, cn()), ret: cn()):
    /*
     * ...
     */
    range(a+1, b, f, ret)
```

CFF-convertible if

- recursion-free or
- tail-recursive

Not in CFF

```
void range(int a, int b, function<void(int)> f) {
    //...
    range(a+1, b, f);
}
```

```
range(a: int, b: int, f: cn(int, cn()), ret: cn()):
    /*
     * ...
     */
    range(a+1, b, f, ret)
```

CFF-convertible if

- recursion-free or
- tail-recursive

Not in CFF

```
void range(int a, int b, function<void(int)> f) {
    //...
    range(a+1, b, f);
}
```

```
range(a: int, b: int, f: cn(int, cn()), ret: cn()):
    /*
     * ...
     */
    range(a+1, b, f, ret)
```

CFF-convertible if

- recursion-free or
- tail-recursive

Not in CFF

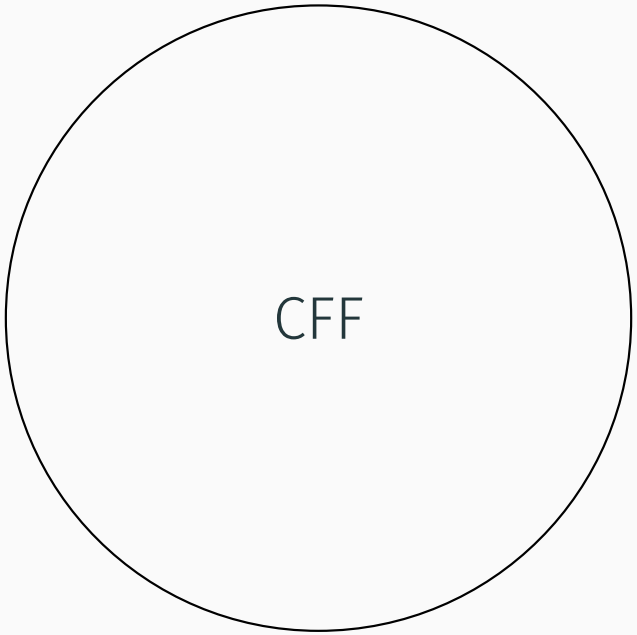
```
void range(int a, int b, function<void(int)> f) {
    //...
    range(a+1, b, f);
}
```

```
range(a: int, b: int, f: cn(int, cn()), ret: cn()):
    /*
     * ...
     */
    range(a+1, b, f, ret)
```

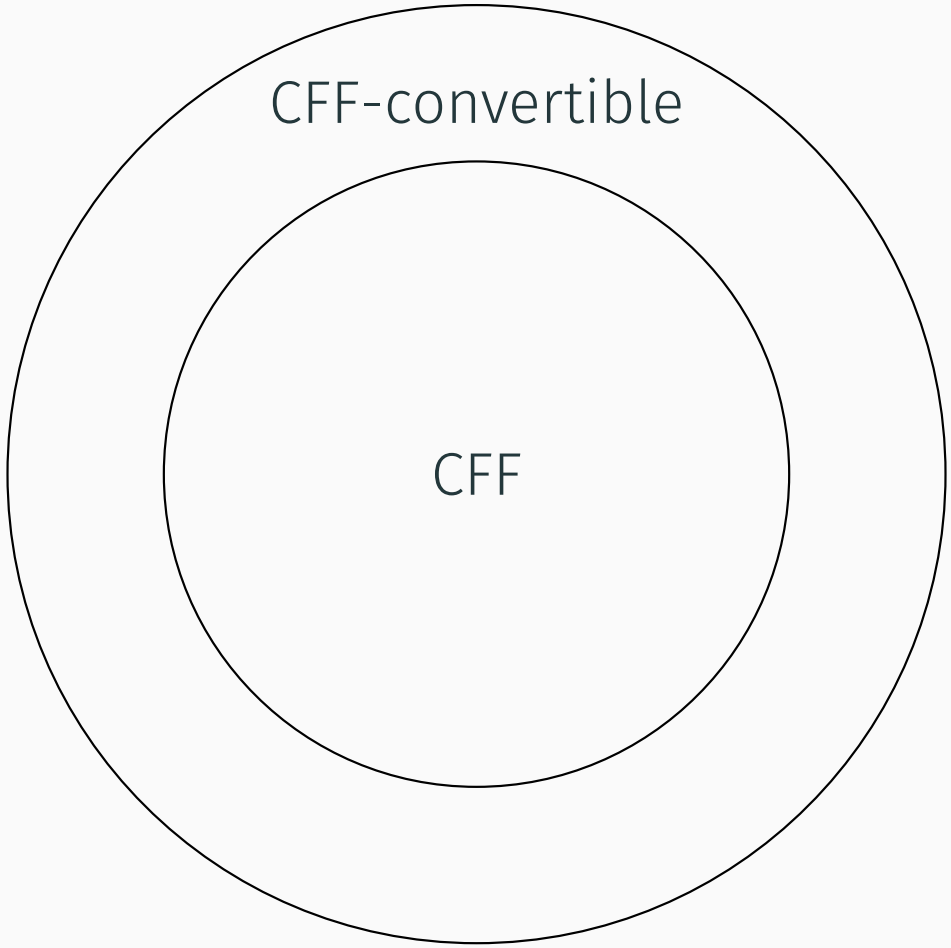
CFF-convertible if

- recursion-free or
- tail-recursive

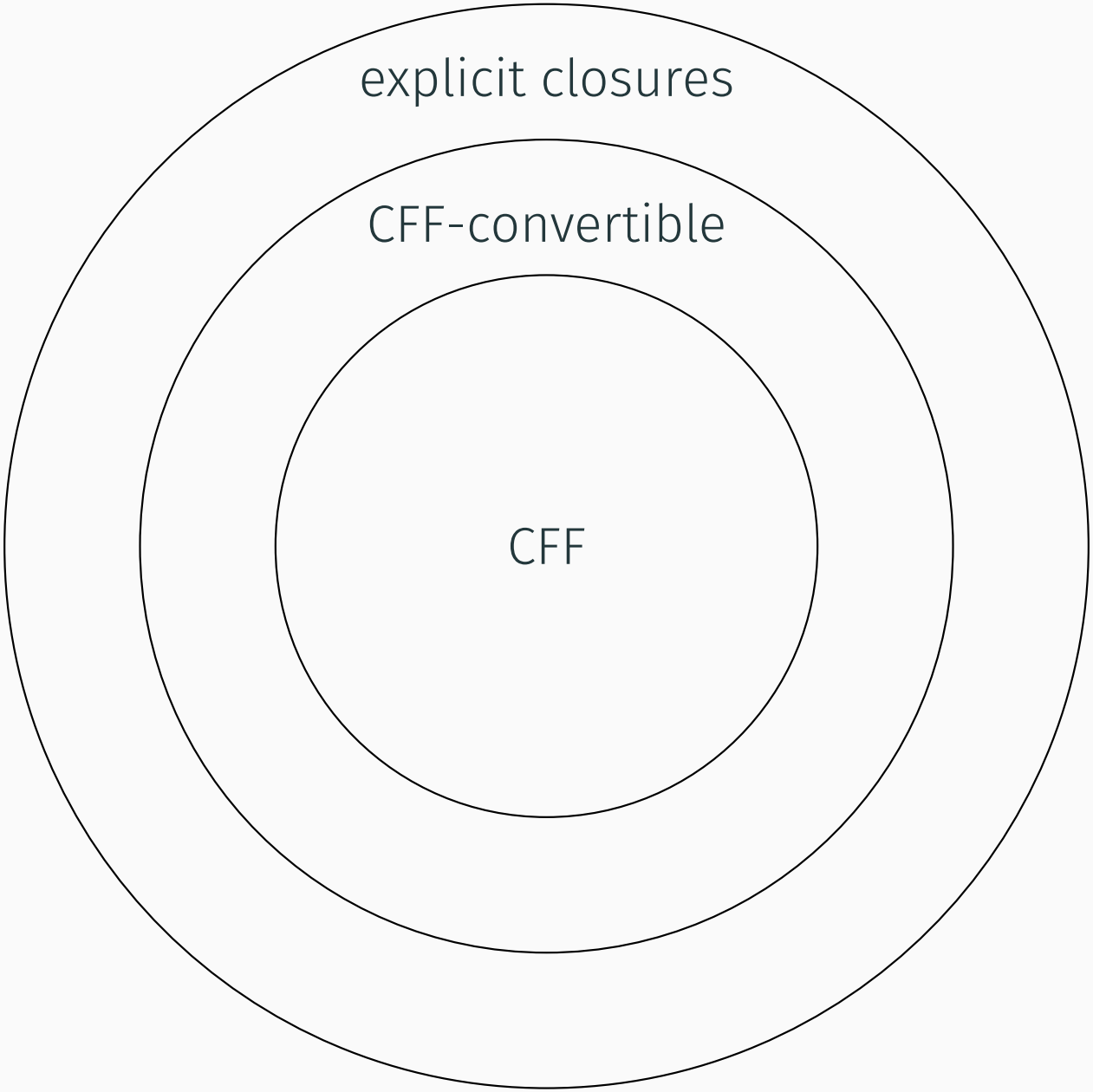
Classes of Thorin Programs



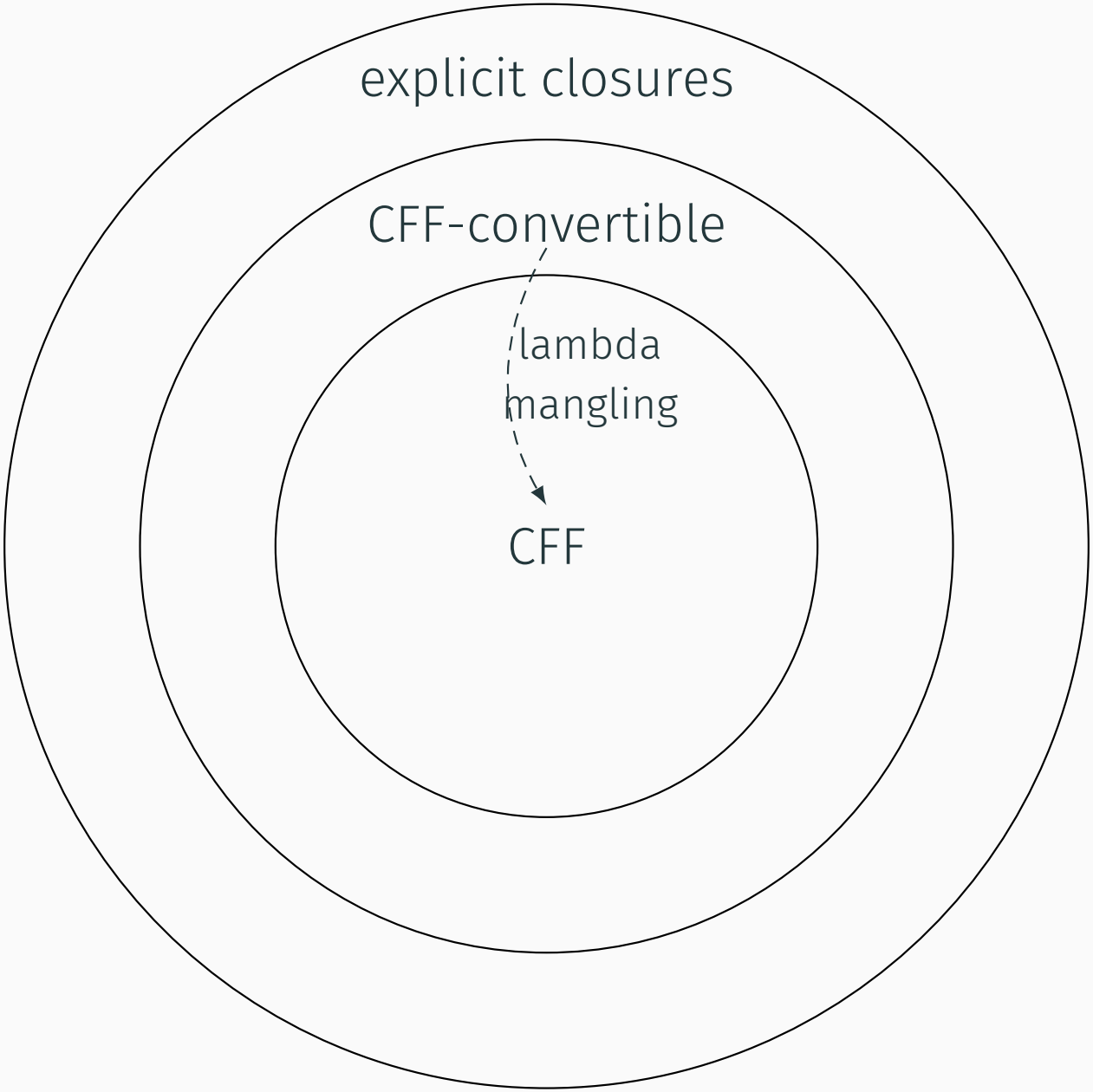
Classes of Thorin Programs



Classes of Thorin Programs



Classes of Thorin Programs



Lambda Mangling = partial inlining/outlining

- (partial) inlining
- (partial) outlining

Lambda Mangling = partial inlining/outlining

- (partial) inlining
- (partial) outlining
- clone basic blocks/functions

Lambda Mangling = partial inlining/outlining

- (partial) inlining
- (partial) outlining
- clone basic blocks/functions
- loop peeling
- loop unrolling

Lambda Mangling = partial inlining/outlining

- (partial) inlining
- (partial) outlining
- clone basic blocks/functions
- loop peeling
- loop unrolling
- tail-recursion elimination
- ...

Impala

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    post_order_visit(n, |d| {
        println(d);
    });
}
```

Impala - `for` Syntax

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    for d in post_order_visit(n) {
        println(d);
    }
}
```

Impala - sum

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    for d in post_order_visit(n) {
        result += d
    }

    println(result);
}
```

Impala - sum

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    for d in post_order_visit(n) {
        result += d
    }

    println(result);
}
```

Impala - sum

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    for d in post_order_visit(n) {
        result += d
    }

    println(result);
}
```

Impala - `return` is the new `continue`

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    post_order_visit(n, |d| {
        if d == 23 {
            return()
        }
        result += d
    }

    println(result);
}
```

Impala - `return` is the new `continue`

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    post_order_visit(n, |d| {
        if d == 23 {
            return()
        }
        result += d
    }

    println(result);
}
```

Impala - `return` is the new `continue`

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    post_order_visit(n, |d| {
        if d == 23 {
            return()
        }
        result += d
    }

    println(result);
}
```

Impala - `return` is the new `continue`

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    post_order_visit(n, |d| {
        if d == 23 {
            return()
        }
        result += d
    }

    println(result);
}
```

Impala - `continue` is the new `return`

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    for d in post_order_visit(n) {
        if d == 23 {
            continue()
        }
        result += d
    }

    println(result);
}
```

Impala - Give me a `break`, please!

```
fn sum(n: &Node) -> () {
    let mut result = 0;

    for d in post_order_visit(n) {
        if d == 23 {
            break()
        }
        result += d
    }

    println(result);
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    for d in post_order_visit(n) {
        println(result);
    }
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    for d in post_order_visit(n) {
        println(result);
    }
}
```

Impala

```
fn post_order_visit(n: &Node, f: fn(int) -> ()) -> () {
    if n.left != nil {
        post_order_visit(n.left, f);
    }
    if n.right != nil  {
        post_order_visit(n.right, f);
    }
    f(n.data)
}

fn print(n: &Node) -> () {
    for d in post_order_visit(n) {
        println(result);
    }
}
```

Generated LLVM (1)

```
define internal void @post_order_visit_392(%Node* %n_394) {
post_order_visit_392_start:
  br label %post_order_visit

post_order_visit:
  %0 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 1
  %1 = load %Node*, %Node** %0
  %2 = icmp ne %Node* %1, null
  br i1 %2, label %if_then, label %if_else

if_then:
  call void @post_order_visit_392(%Node* %1)
  br label %next

if_else:
  br label %next
; ...
```

Generated LLVM (1)

```
define internal void @post_order_visit_392(%Node* %n_394) {
post_order_visit_392_start:
  br label %post_order_visit

post_order_visit:
  %0 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 1
  %1 = load %Node*, %Node** %0
  %2 = icmp ne %Node* %1, null
  br i1 %2, label %if_then, label %if_else

if_then:
  call void @post_order_visit_392(%Node* %1)
  br label %next

if_else:
  br label %next
; ...
```

Generated LLVM (1)

```
define internal void @post_order_visit_392(%Node* %n_394) {
post_order_visit_392_start:
  br label %post_order_visit

post_order_visit:
  %0 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 1
  %1 = load %Node*, %Node** %0
  %2 = icmp ne %Node* %1, null
  br i1 %2, label %if_then, label %if_else

if_then:
  call void @post_order_visit_392(%Node* %1)
  br label %next

if_else:
  br label %next
; ...
```

Generated LLVM (1)

```
define internal void @post_order_visit_392(%Node* %n_394) {
post_order_visit_392_start:
  br label %post_order_visit

post_order_visit:
  %0 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 1
  %1 = load %Node*, %Node** %0
  %2 = icmp ne %Node* %1, null
  br i1 %2, label %if_then, label %if_else

if_then:
  call void @post_order_visit_392(%Node* %1)
  br label %next

if_else:
  br label %next
; ...
```

Generated LLVM (2)

```
; ...
next:
  %3 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 2
  %4 = load %Node*, %Node** %3
  %5 = icmp ne %Node* %4, null
  br i1 %5, label %if_then2, label %if_else1

if_then2:
  call void @post_order_visit_392(%Node* %4)
  br label %next3

if_else1:
  br label %next3

next3:
  %6 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 0
  %7 = load i32, i32* %6
  call void @println(i32 %7)
  ret void
}
```

Generated LLVM (2)

```
; ...
next:
  %3 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 2
  %4 = load %Node*, %Node** %3
  %5 = icmp ne %Node* %4, null
  br i1 %5, label %if_then2, label %if_else1

if_then2:
  call void @post_order_visit_392(%Node* %4)
  br label %next3

if_else1:
  br label %next3

next3:
  %6 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 0
  %7 = load i32, i32* %6
  call void @println(i32 %7)
  ret void
}
```

Generated LLVM (2)

```
; ...
next:
  %3 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 2
  %4 = load %Node*, %Node** %3
  %5 = icmp ne %Node* %4, null
  br i1 %5, label %if_then2, label %if_else1

if_then2:
  call void @post_order_visit_392(%Node* %4)
  br label %next3

if_else1:
  br label %next3

next3:
  %6 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 0
  %7 = load i32, i32* %6
  call void @println(i32 %7)
  ret void
}
```

Generated LLVM (2)

```
; ...
next:
  %3 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 2
  %4 = load %Node*, %Node** %3
  %5 = icmp ne %Node* %4, null
  br i1 %5, label %if_then2, label %if_else1

if_then2:
  call void @post_order_visit_392(%Node* %4)
  br label %next3

if_else1:
  br label %next3

next3:
  %6 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 0
  %7 = load i32, i32* %6
  call void @println(i32 %7)
  ret void
}
```

Generated LLVM (2)

```
; ...
next:
  %3 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 2
  %4 = load %Node*, %Node** %3
  %5 = icmp ne %Node* %4, null
  br i1 %5, label %if_then2, label %if_else1

if_then2:
  call void @post_order_visit_392(%Node* %4)
  br label %next3

if_else1:
  br label %next3

next3:
  %6 = getelementptr inbounds %0, %Node* %n_394, i32 0, i32 0
  %7 = load i32, i32* %6
  call void @println(i32 %7)
  ret void
}
```

Evaluation

Benchmarks – The Computer Language Benchmark Game[1]

	runtime in ms	
	C	Impala
aobench	1.220	1.357
fannkuch-redux	27.137	28.070
fasta	2.313	1.517
mandelbrot	2.143	2.113
meteor-contest	0.047	0.043
n-body	5.497	6.130
pidigits	0.710	0.763
regex	6.477	6.470
reverse-complement	1.090	1.220
spectral-norm	4.423	4.480

[1] https://benchmarksgame.alioth.debian.org/

Benchmarks – The Computer Language Benchmark Game[1]

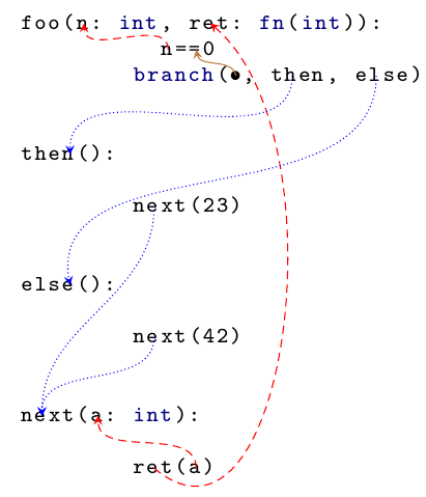
	runtime in ms	
	C	Impala
`aobench`	1.220	1.357
`fannkuch-redux`	27.137	28.070
`fasta`	2.313	1.517
`mandelbrot`	2.143	2.113
`meteor-contest`	0.047	0.043
`n-body`	5.497	6.130
`pidigits`	0.710	0.763
`regex`	6.477	6.470
`reverse-complement`	1.090	1.220
`spectral-norm`	4.423	4.480

- high-order IR does not “hurt” performance
- all closures removed

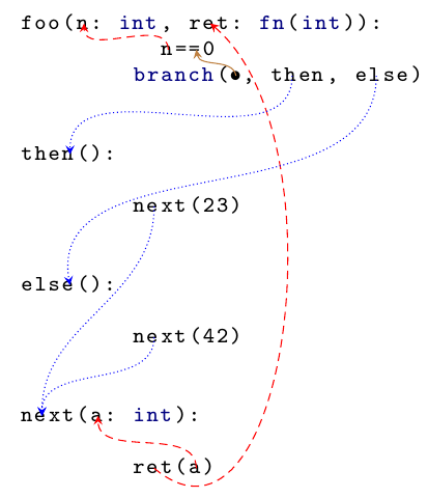
[1] `https://benchmarksgame.alioth.debian.org/`

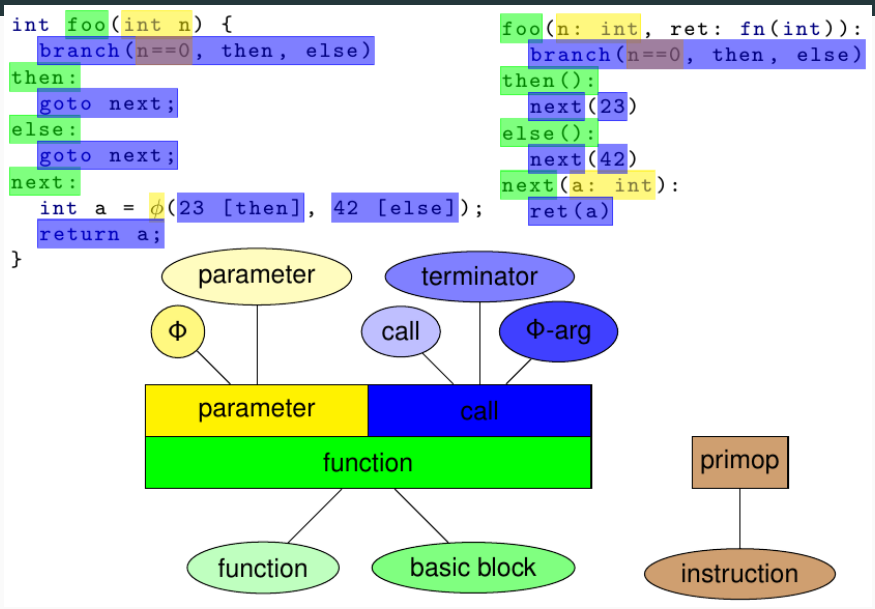
Summary

Conclusions

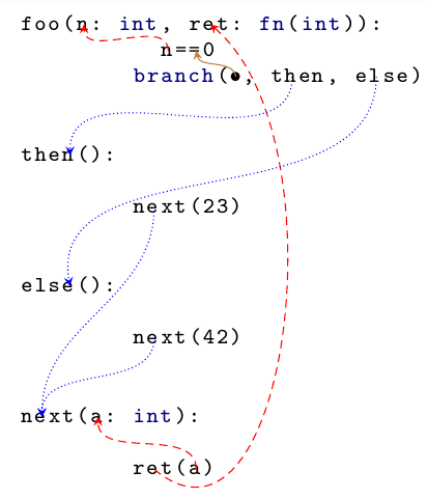


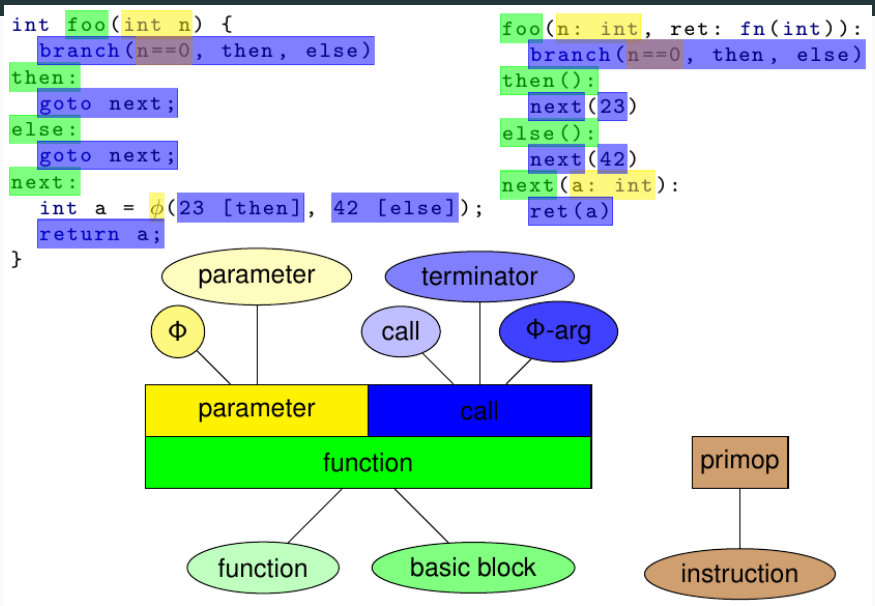
Conclusions

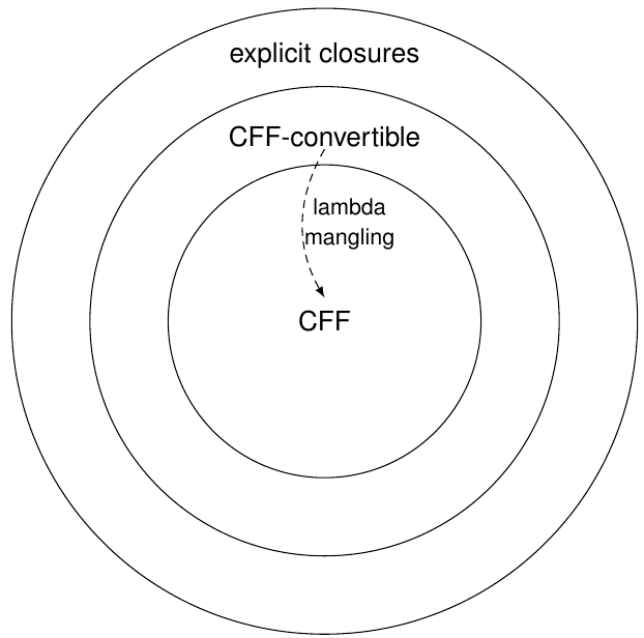




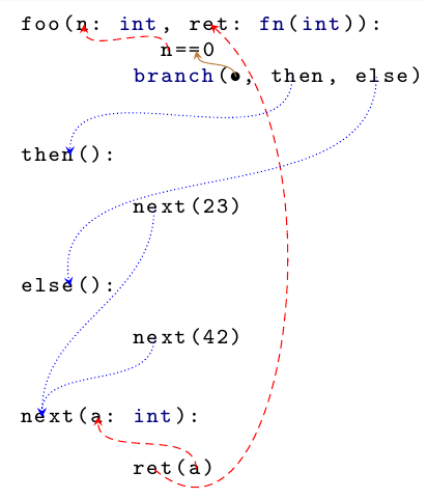
Conclusions

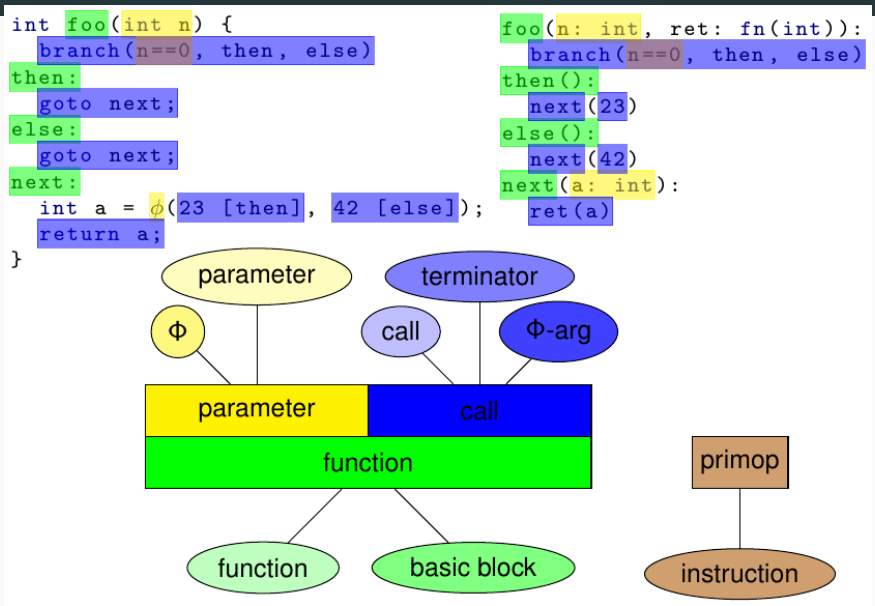


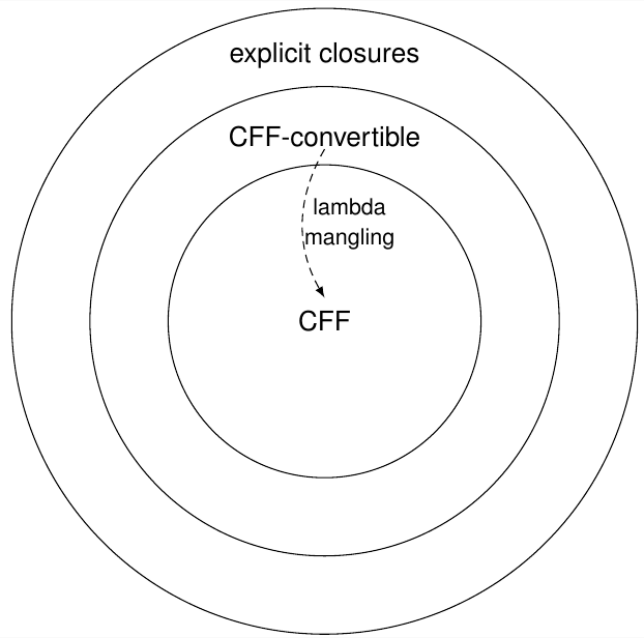




Conclusions







Thank you!

Questions?